Discrete-time linear control
over CSMA/AMP network

Marek Peca
FEL ČVUT

supervisors:
Guy Juanole
Gérard Mouney

LAAS CNRS       June 2005
Discrete-time linear control over CSMA/AMP network

Marek Peca
FEL ČVUT

supervisors:
Guy Juanole
Gérard Mouney

LAAS CNRS June 2005
mílemu dědečkovi
## Contents

1 Introduction ................................................. 5

2 Problem specification .................................. 6

3 Experimental system configuration .................. 8
   3.1 Microcontroller realization ..................... 8
   3.2 Simulation in TrueTime .......................... 9
   3.3 Plant used in experiments (motor) ............... 10

4 Treatment of network inside control loop .......... 12
   4.1 FIFO buffering .................................. 12
   4.2 Buffering with old value replacement .......... 13
      4.2.1 Comparison of FIFO and value replacement approach .......... 15
      4.2.2 Hardware implementation of value replacement .............. 17
      4.2.3 Value replacement in TrueTime .......................... 17
   4.3 Irregular sampling and no variable delay ........ 17
      4.3.1 Hardware implementation of irregularly sampled control system ........ 19
      4.3.2 Irregular sampling in TrueTime ...................... 20

5 Controller design ....................................... 21
   5.1 Ordinary discrete controller .................... 21
      5.1.1 Controllers used in experiments .................. 21
   5.2 Controller adapted for distributed system ...... 22
      5.2.1 Irregularly sampled control system composition ...... 23
      5.2.2 Irregularly sampled controller design ............. 24
         5.2.2.1 Irregularly sampled observer .................. 25
         5.2.2.2 Treatment of optional variable delays .......... 26
         5.2.2.3 Controller with irregularly sampled static state feedback .......... 26
      5.2.3 Note about “compensation approach” ............. 27
      5.2.4 Stability of irregularly sampled state feedback controller .......... 28
      5.2.5 Irregularly sampled static state feedback controller design .......... 28

6 Dynamical priorities .................................. 34
   6.1 Motivation ........................................... 34
   6.2 Message priority implementation ................. 34
   6.3 Sensor-controller message priority ............... 35
   6.4 Distribution of environment message priorities ..... 35
   6.5 Environment traffic simulation ................... 36
      6.5.1 Implementation .................................. 37
      6.5.2 Bit stuffing difficulty .......................... 38
   6.6 Message priority computation ..................... 38
      6.6.1 Error proportional priority ..................... 39
      6.6.2 Priority proportional to filtered error ............ 41
   6.7 Problem of solution-quality criteria ............. 43
   6.8 Dynamical priorities with controller adapted for distributed system .......... 43

7 Conclusion ............................................. 45

8 Bibliography ............................................ 46
Acknowledgements

I would like to thank to my parents, Jaroslav and Darina, for my whole life, upbringing and support. I would like to thank to my grand-father Ing. Miloslav Hrala, who unveiled for me much of the world and nature and who led my heart to science and engineering.

Next, I would like to thank to mademoiselle Lucie Žáková, who gave me an idea of going to Toulouse, where this work has been done. I would like to thank to my teacher and director Doc. Zdeněk Hanzálek in Prague and Prof. Guy Juanole in Toulouse, who made my Toulouse dream possible, even at cost of considerable administrative efforts.

Along with Prof. Guy Juanole, I would like to thank to Prof. Gérard Mouney, Christophe Calmettes and Frédéric Gouaisbaut for their help with this work and for answering my questions.
1 Introduction

This work is concerned with networked control systems. Control systems, developed on a base of linear system theory, composed from several parts, connected together by a computer network, are of interest. Goal of the work was to observe such distributed control system behaviour under conditions, where the network is shared with other unaffiliated tasks, and examine influence of network traffic load to control system performance and stability.

My task was to implement the distributed control system with real controlled plant (servomotor) using CAN factory standard bus and available C167 microcontroller computer boards, and examine operation of discrete-time controllers in this control system. For the second, I had to implement method of dynamical priority assignment, which should improve control system performance under high network traffic load.

Beside real implementation, various network control system simulations have been performed using TrueTime software package for Matlab and Simulink suite. Other computations and design has been performed using Matlab as well. Microcontrollers C167 have been used for CAN network implementation and programmed using HighTec GNU C compiler for C16x processor family.

Treatment of network from control system point of view has been revisited and also use of irregularly sampled controller with independent sampling of sensor and actuator signals has been proposed, along with design and stability proof directions, and tested in simulation.

Program source codes and data, belonging to this work, are available at <http://duch.cz/mp/laas/can_control/>.

Contact e-mail address for questions and comments is <marek.peca@duch.cz>.
2 Problem specification

This work deals with digital discrete control systems, distributed into several parts, connected together by computer network. The control system is attached through the network to controlled plant at two points – actuator output (plant input) and sensor input (plant output) and has some source of reference value – either software generated, measured or user entered. Control system regulates the plant to adjust its output to approach (or, more generally, to follow) the reference value, through the use of feedback of plant output value compared to the reference.

By distributed control system, we mean a set of two or more nodes, containing sensor, actuator and optionally reference input, connected by a network, where the control feedback loop has at least one point, where values travel between nodes by the network (the loop is closed over the network).

The common discrete control system [1] is based on periodic sampling of values. The difference of distributed control system from ordinary discrete control system is, that it is not possible at any time to transport a network message (value in some point of loop) immediately, because the network medium is shared with other users (network environment).

Among several strategies to shared media access, we focused on CSMA/AMP\(^2\) type network. This access strategy allows to influence precedence of messages from concurrent nodes, and is also widely implemented in industrial applications [\[\rightarrow 3.1\]].

On CSMA/AMP type network, the time, for which node can not transmit, mainly depends on relative message priority of the node and concurrent senders. There still can be an access delay caused by message of lower priority, if its transfer is already in progress (effect called priority inversion). But this delay can be neglected, if the network traffic load is high, in comparison with delay caused by other higher-prioritized messages, waiting or generated in the same moment in other nodes.

---

1. abbreviation of “discrete-time”, i.e. working in time steps; however, this need not to mean strictly periodic sampling
2. Carrier Sense Multiple Access with Arbitration by Message Priority, also known as CSMA/BA (Bit Arbitration)
This chronograph shows an example of message processing on CSMA/AMP network (simulated by software package TrueTime, [→3.2]). Each curve corresponds to particular network node. If the line is low, the node does not want to transmit anything. If it is in the middle, the node wants to transmit, but its message must wait for media access, and if it is high, the message is currently being sent over the network. In this case, the node A sends messages with highest priority, node B with medium, and C with lowest priority.

The time, for which the node of control system needs to wait before sending, depends on coincidence of discrete control timing and other processes on the network. If messages of control system do not have the highest priorities, we cannot achieve constant time period between control system steps, what is assumed in ordinary discrete control design. Also, it is very difficult, or impossible, to predict exactly actual time intervals between messages.
3 Experimental system configuration

For our experiments and simulation, distributed system consisting of three network nodes – sensor, actuator and controller node, has been chosen. Among traffic of network environment, two types of messages are transmitted to the network – message from sensor to controller, called later a sensor-controller message, and message from controller to actuator, called an controller-actuator message.

Our work has been focused on regulation of output to constant reference value. Where it makes difference, we do not bother with control tasks of higher order, such as following of reference with constant derivative\(^3\). Actuator sets output value immediately after its reception from the network in zero-order hold mode, ie. the output is held constant until next value arrival. In this setup, optional network transport of reference value, which has to be present in controller node, is not subject of study – in the task of regulation to a constant value it does not have an importance.

3.1 Microcontroller realization

Primarily, the experimental distributed control system has been build on a CAN\(^4\) network [3]. CAN, specifying physical and link network layers, offers arbitration based on 11 bit (version 1.0) or 29 bit (version 2.0B) numbers, which also serve as unique message identifiers. Each message can carry up to 8 bytes of information. In all our applications, 11 bit identifiers and messages up to 32 data bits has been used. CAN network speed in all presented experiments was 50000 bit s\(^{-1}\).

Each network node has been implemented in a computer board [4], based on Siemens (now Infineon) C167 microcontroller [5]. C167 contains hardware CAN controller with several intelligent buffers and automatic incoming message filtering. C167 has a 16 bit processing unit core and has been programmed using GNU C

---

\(^3\) somewhere called “servo problem”

\(^4\) Controller Area Network, automotive industrial standard introduced by Bosch company
compiler **gcc** and debugger **gdb**, ported to C16x microcontroller family by HighTec\(^5\) company\(^6\). This platform offered reasonable floating-point processing and networking capabilities for our task.

The experimental system consists of four microcontroller network nodes – sensor, controller, actuator and traffic generator, simulating a network traffic of the network environment by intensive sending of meaningless messages. The actuator and sensor are connected to the plant, which in our case is a motor, described later [\(\rightarrow\) 3.3]. Actuator is interfaced to analog input of motor amplifier by onboard 12 bit DAC\(^7\), sensor measures position potentiometer value by C167 integrated 10 bit ADC\(^8\).

Beside its control operation, sensor node measures plant output value, in finer granularity than it is used for regulation, and also listens to network messages from controller to actuator (allowed by CSMA principle). These data of plant input and output are sent by asynchronous serial line to UNIX workstation for further processing or visualisation. An interactive text terminal is attached by another asynchronous serial line to traffic generator, for adjusting of online generated traffic parameters.

### 3.2 Simulation in TrueTime

Beside the real hardware implementation, simulations in specialized software package TrueTime have been performed. TrueTime\(^9\) is a package, integrated into Matlab/Simulink software system, and dedicated to allow

---

\(^5\) [http://www.hightec-rt.com/c16x.html](http://www.hightec-rt.com/c16x.html)

\(^6\) however, their commercial policy violated GNU rules and status of their port is unknown

\(^7\) Digital to Analog Converter

\(^8\) Analog to Digital Converter

\(^9\) developed at Department of Automatic Control of Lund Institute of Technology.
simulation of computer networks and multiprocessing kernels\textsuperscript{10}, interconnected with dynamical systems. Each kernel can be interconnected with traditional Simulink blocks (eg. models of dynamical systems) and with the network. All events, network and interrupts, are handled externally by Simulink as ordinary real-valued signals. Simulation is event-driven and based on Simulink. In each step of simulation, some code in kernels can be executed. The code, representing simulated processes, can be written in Matlab M-language or in C++.

\[
\begin{align*}
\text{u, y, r} & \quad \text{Snd} \\
\text{A/D Snd} & \quad \text{sensor} \\
\text{traffic load} & \\
\text{A/D} & \quad \text{Rev} \\
\text{controller} & \\
\text{Schedule} & \quad 1 \\
\text{TrueTime Network} & \\
\text{A/D} & \quad \text{Snd} \\
\text{Rcv} & \quad \text{Snd} \\
\text{Rcv} & \\
\text{D/A} & \quad \text{saturation} \\
\text{9.09} & \quad 0.23s^{+} + s \\
\text{motor} & \\
\text{A/D} & \quad \text{Rev} \\
\text{Actuator} & \\
\text{Rcv} & \quad \text{Snd} \\
\text{Schedule} & \\
\text{1} & \quad \text{trueTime Network} \\
\end{align*}
\]

\textit{distributed control system simulation within TrueTime/Simulink}

In our TrueTime simulations, also the three control system nodes (kernels) and one traffic generator have been established. The same plant as in reality [\cite{3.3}] has been modelled. The CSMA/AMP block has been run eight-times faster and message lengths were entered eight-times longer\textsuperscript{11}, however, this is not sufficient for precise CAN simulation, see \cite{6.5.2}. Network speed in all simulations was 50000 bit s\textsuperscript{-1}. Code of nodes has been written in M-language, one additional function (in C++) has been added into TrueTime core \cite{4.2.3}.

3.3 Plant used in experiments (motor)

As the plant, used in experiments and simulation, a DC\textsuperscript{12} motor in positional mode (servomotor) has been used. Model of this motor has been given as \(P(s) = \frac{K}{s(1 + Ts)}\) (Laplace transform of impulse response),

\textsuperscript{10} representing computers with interrupt and real-time capabilities
\textsuperscript{11} in present version, TrueTime allows entering of message lengths in whole bytes only
\textsuperscript{12} Direct Current, brushed
where gain $K = 9.09\pi \text{rad V}^{-1}\text{s}^{-1}$ and time constant $T = 0.23 \text{s}$, see simulation diagram on following picture.

With these numbers, the input is voltage in volts and output is number of whole turns. Input voltage is in range $(-5, 5)\text{V}$, what determines actuator saturation. Also a friction has been observed, in few simulations modelled as insensitivity area $(-0.3, 0.3)\text{V}$ (not much well fitted model). A single-turn potentiometer senses position in almost whole-circle range. As a control error value in the controller, the smaller of two angles is used, i.e. the absolute value of error is never greater than half a turn.

For ordinary discrete controller design methods, a common period of approximately $T_0 = 0.025 \text{s}$ has been used for model discretization in almost all of our experiments.
4 Treatment of network inside control loop

Although single kind of network, the CSMA/AMP, has been selected, there are still different ways, how to treat the network in distributed control application. It means to specify behaviour of transmitting node in moment, when network is transferring other, more prioritized messages.

4.1 FIFO buffering

With FIFO\textsuperscript{13} pipe-like approach, the value of sample, when measured or computed, is put into queue and when the network becomes available, the oldest sample from the queue is sent. This behaviour is common mainly in high-level network applications, rather than in world of industrial CSMA/AMP networks and microcontrollers.

This approach places a random delay on each sample. If the network load is so high, that delays longer than time-distance between samples can appear, then the delay will be cumulative: if sample $S_1$ is delayed by $\tau_1$ and sample $S_2$ was produced in time $T$ after $S_1$ and $\tau_1 > T$, then delay of $S_2$ can be regarded as $\tau_2 + \tau_1 - T$, where $\tau_2$ would be a value of $S_2$ delay, if the queue is empty. Use of queue also requires the network being able to transfer all generated messages in reasonable time, otherwise the queue length will grow to infinity.

Following chronograph shows delays, present with FIFO strategy at controller. Meaning of curves is the same as in $\langle\leftarrow\rightarrow\rangle$ – request of message transfer by particular node is indicated by rising edge from bottom to middle position, during middle position the message is waiting and scheduled after next rising edge to top position. Finished message transmission is indicated by falling edge. Delays, induced by network load, generated in node $E$, between messages of sensor node $S$ and controller node $R$, are indicated by arrows. These delay values are also plotted in graph in comparison $\langle\leftarrow 4.2.1\rangle$.

\textbf{sample delays in case of FIFO buffering}

\textsuperscript{13} First In First Out – principle of queue
Sample delay is a random variable, whose mean value can vary very much. Although all sample values are preserved, their time-position is uncertain. Maybe this delay could be estimated and took into account in controller, but for such more complicated control system I would rather recommend other method of network treatment \[\rightarrow 4.3\]. The cumulative property of introduced delays are the main reason, why this approach does not work well in distributed control system.

### 4.2 Buffering with old value replacement

In the most natural and simplest implementation, measured or computed sample is put into single-value buffer\(^\text{14}\), from which it is sent upon network release. If new sample arrives before the old was sent, the old sample is discarded and replaced by the new (intentional sample erasure).

This behaviour would be exactly same as with FIFO-queue, if the delay is always less than intersample time-distance. Otherwise, sample erasures will appear and remaining samples are delayed no more, than by length of corresponding intersample period. This approach has been used in most of our work and has shown significantly better results than FIFO-queue. Delays are shown on following chronograph, which meaning is same as above, and sample erasures can be seen as few missing arrows at falling \(S\) edge. The same values are also in comparison graph \[\rightarrow 4.2.1\].

![Chronograph showing sample delays in case of old value replacement](image.png)

**sample delays in case of old value replacement**

Problem of sample erasure does not have noticeable impact on controller-actuator messages. If we would rather preserve the value (as with FIFO-queue), then after network release, more messages will be sent within shortened time (otherwise the condition of average capacity is not met and buffer will be overfull). Thus, the values before the last one are actuated for a short time, what means, their integral effect at zero-order hold has a little significance. This situation is illustrated in following figure – network was busy during actuation of sample 1 and then, in first case, samples 2 and 3 have been actuated, in second case erased.

\(^{14}\) buffer is single only in respect to the application layer – for proper operation, the underlying hardware or software of network interface should implement necessary double- or triple-buffering
The sample erasure has an effect on sensor-controller messages. If an ordinary discrete controller ([\[\rightarrow 5.1\]]) is used, new sample, substituted in place of missing one, will have particular influence on parts of controller computation. Let us make a short consideration about classical discrete PID controllers. For a proportional component, similar assumption as with controller-actuator message can be done. Empirically, it is better to reflect fresh, than obsolete value. For an integral action, a value is missing, so the action is diminished. However, this should be only a little fraction of whole cumulative sum. The largest impact will have the missing sample on derivative action. If sampling a signal with constant derivative (velocity), difference will be twice bigger with one sample erasure, as it is shown on following figure. But, this distortion should be comparable with effect of variable sample delay itself.

\[\text{little impact of shortly actuated values at zero-order-hold}\]

Proportional, Integral, Derivative, or PSD – Proportional, Summation, Difference
4.2.1 Comparison of FIFO and value replacement approach

Comparisons between old value replacement and FIFO-queue have been made and replacement approach shown better results for all our discrete controller designs. By better results we mean, the system has been stabilized in shorter time, or the system remained stable in contrary to the other under same network conditions. By network conditions we mean, the environment traffic network load had the same statistical properties.

Following graphs show simulated transient responses for control systems using FIFO and value replacement strategies. In both cases, the same environment traffic generation has been used, set to 87% of network capacity with regularity $r = 0.94$ (see \cite{6.5}). Sensor-controller and controller-actuator messages has been of length 66 bits, environment messages of 50 bits. PD controller has been used \cite{5.1.1}. 

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig.png}
\caption{Jitter in computed difference caused by sample erasure}
\end{figure}
transient response of system using value replacement strategy (stable)

transient response of system using FIFO-queue (unstable)

Analyzed sample delays from these two simulation events are plotted in following graph. These are the same values, as depicted by chronograph arrows in previous sections. Results have been obtained by TrueTime simulation.
4.2.2 Hardware implementation of value replacement

Strategy of value replacement is the most natural way of implementation in microcontrollers, or more generally in all low-level hardware design (e.g., FPGA). Only few buffers (or registers) for CAN implementation are needed, one as part of serializing shift-register, second for waiting data, and third for CPU (task) write access. In our realization with C167 microcontroller containing hardware CAN controller, the network functionality is provided in parallel to CPU core execution and the network message buffer (so called message object) contents can be safely rewritten by new value anytime.

4.2.3 Value replacement in TrueTime

In present version of TrueTime, only the FIFO strategy has been implemented. To enable more realistic obsolete value replacements, function ttDiscardUnsentMessages has been written and incorporated into TrueTime simulation kernel. Almost everything has been left as is, only upon call of this function, the network message queue for given (function caller) node is examined, and all messages, which are not yet being transferred, are discarded. For simulation with value replacement approach, the function ttDiscardUnsentMessages is called immediately before putting of new value to message buffer by ttSendMsg.

4.3 Irregular sampling and no variable delay

In previous treatment of network in control loop, two types of delay are present. First one is a delay, consisting of time of sample data transfer at given speed, including some protocol overhead and processing time in nodes. This delay is usually constant and often very short, thus it can be neglected, or incorporated...
easily into system model for control design. The second, *variable* delay, is induced by network environment traffic, causing the network is not available for transfer of control loop samples in some time intervals. Presence of this hardly predictable delay does a problem.

On the other hand, there is actually *no variable delay* during the network transfer – there are only random time intervals, within whose the particular control system node can not transmit. If we would be able to measure and compute sample value in the exact moment, when network becomes available, we will get network transfer with *relatively short* and *constant delay*, but with *irregular sampling*\(^{18}\).

In such irregularly sampled control system without variable delays, we measure (sample) value at sensor immediately before sensor-controller message sending, and compute command value immediately before controller-actuator message sending. Operation of the system is shown in following figure. Two chronographs of network flow are shown, \(S\) for sensor-controller messages and \(R\) for controller-actuator messages. In upper part, measured plant output is sampled when sending sensor-controller messages. In lower part, *on-the-fly* computation of desired command value is performed all the time and it is sampled when sending controller-actuator messages, and these values are held at actuator. (Plotted data are from simulation, using plant without input saturation.)

\(^{18}\) also known as “non-uniform sampling” – time intervals between samples are not constant period, as assumed in most of discrete control and signal theory assumptions.
It may be a problem to measure and especially to compute the value in so short time of the network availability detection. It is possible to do both tasks in repeated loops and to refresh periodically the value in buffer, waiting for network transmission. This will introduce random variable delay between zero and the period of loop, i.e. the calculation or measurement duration. By speed of these operations, this random delay could be made negligible.

If the time of measurement or computation is not short enough, a compromise solution could be done by running a counter in parallel. The counter would make a timestamp, counting from last value update (measurement or computation) – the message could be dated. With this strategy, an additional variable, but known delay is measured by timestamp, thus can be written into sensor-controller message. It makes no sense to write it into controller-actuator message, but it can be useful at the controller node for better knowledge of precise time history of actuated values. In this approach, the known variable delay exists together with irregular sampling, what makes it slightly more complicated, than with irregular sampling without the delay, but controller design considerations are similar for both.

Of course, variable intersample time intervals must satisfy some requirements, empirically they must not be longer than some period or they must lie in some range of values. This is common need in any approach to distributed control system design. Note, that irregularity of the sample arrival appears in previous methods, FIFO and value replacement, as well, but in opposite to irregular sampling strategy, it is not taken into account (is not measured). Following graphs show difference between regular sampling with variable delay and optional erasure (upper graph), and irregular sampling (lower graph).

![Graph showing regular sampling with delay vs. irregular sampling](image)

\[ y \]

\[ t [s] \]

regular sampling + delay (erasures) vs. irregular sampling

\[ ^{19} \text{eg. by microcontroller timer circuitry and interrupt} \]
4.3.1 Hardware implementation of irregularly sampled control system

In hardware CAN controller implementations, there always exists a flag or interrupt, letting to know, when the sending of message succeeded. As stated above, the measurement (eg. from ADC or timer/counter) and computation of controller output is to be done in fast loop, replacing sample value in the buffer and optionally resetting timestamp upon new value placement. After network becomes available, the node is acquainted by an interrupt or by checking CAN transmission flag.

Thus, the controller node has an information about time position of sample measurement and actuation of new value. Both controller and sensor node should count number of sent messages and adapt sending of new messages to maintain optimal average frequency, and to keep intersample periods in required bounds. Note, that period bounds or average can be different for sensor and controller-actuator messages.

Hardware implementation of irregular sampling approach have not been realized during this work.

4.3.2 Irregular sampling in TrueTime

Facility of buffer value replacement has been added by function \texttt{ttDiscardUnsentMessages}. To implement irregularly sampled control system, notification about successful message transmission is needed. In TrueTime, each kernel block, implementing network node in our application, has two event inputs – external interrupt (Interrupts on original block, in our schematics IRQ) and network reception notification (Rcv). Events are implemented as change of value of this scalar in Simulink. To get notification upon message transmission, it is possible to connect external interrupt input of transmitting node in parallel with network notification input of receiving node, as it is shown at following simulation diagram.

Continuous-time blocks can be built in Simulink\textsuperscript{20} to implement controller algorithms, so the cyclic recomputation of values will be made by Simulink differential equation solver.

\footnotesize
\textsuperscript{20} as custom so-called S-functions, possibly interconnected with standard Simulink continuous-time blocks

\begin{center}
\textit{finished transmission notification in TrueTime}
\end{center}
5 Controller design

In this work, we focused on \( LTI \) systems, although linearity is always imperfect at plant. The time-invariance is a philosophical question in domain of controllers in distributed systems, when their output is a function of network events, such as sampling irregularities or delays. From one definition, such controller can be still time-invariant, if we treat these network influences as an input. However, in other literature ([9]), such system with network-dependent parameters is called to be time-variant.

5.1 Ordinary discrete controller

By an ordinary discrete controller we mean discrete-time \( LTI \) system, based on constant sampling period \( T_0 \) between samples in relation to continuous-time controlled system. This controller has only input of reference and sensed value and an output of actuated value, ie. no additional information, such as network observation, is present. Input/output relations can be described by polynomial fractions in \( z\)-domain.

Ordinary discrete controller can be used together with described FIFO or value replacement approach. The timing source of period \( T_0 \) is in the sensor node, working in time-triggered mode [8]. Plant output is sampled with period \( T_0 \), and samples are sent in sensor-controller messages to controller node. Controller node, working in event-triggered mode [8], computes each new step upon arrival of sensor sample, and sends computed value in controller-actuator message. Actuator, also working in event-triggered mode, immediately actuates the value. This approach has been used in our experiments and simulations. Alternatively, the timing source can be in controller node, when sensor node listens to controller-actuator messages and sensing of input is thus event-triggered. This approach has not been tested during this work.

However, in given networked environment, assumptions of ordinary discrete control design are not satisfied. Random sample delays, optionally also erasures, are introduced, which can cause control performance degradation, even system instability, depending on controller robustness. Destabilization of control loop due to network induced delays is described in [7]. Although stability of ordinary discrete controller can not be guaranteed generally in described distributed system, I have been asked to test its operation and to try to improve performance using dynamical priorities, see [\( \leftrightarrow 6 \)].

5.1.1 Controllers used in experiments

For our experiments, among classical PD\(^{22}\) controller, an RST controller\(^{23}\) with separate filtration of reference and feedback value, has been used.

PD controller design has been performed many times with similar results, using methods of pole-placement as well as continuous-time controller discretization, finally tuned by hand to adapt overall response to motor nonlinearities – input saturation and friction.

RST controller has been designed for minimal order with no zero-pole cancellations, according to [2]. Plant with transfer function \( P(s) = 9.09/s(1 + 0.23s) \) has been discretized by sampling period \( T_0 = 0.025s \) as \( P(z) = 0.011915(z + 0.9644)/(z − 1)(z − 0.897) \). Desired model of system has been chosen as \( F_w = B^-/z^2 = \)

\(^{21}\) Linear and Time-Invariant, ie. described by linear differential or difference equations with constant parameters and with the time as independent variable

\(^{22}\) PID with integration component zero

\(^{23}\) also known as “two-degree-of-freedom controller”, controller with precorrection or controller branched into feedback and feedforward
(z + 0.9644)/z^2, observer factor as \( L^- = z + 0.1 \). RST controller is described according to following picture by transfer functions \( M(z) = 38.9852 + 3.89852z^{-1} \), \( N(z) = 95.52 - 52.66z^{-1} \), \( T(z) = 1 + 0.7431z^{-1} \).

Experimental transient response has been measured. Graph shows this response along with reference steps, in this case measured from analog source, and command signal. Here the controller operates without influence of network environment (no traffic load).
5.2 Controller adapted for distributed system

The goal is to design a controller, which would take sample transport through the network into account. Design idea of controller, adapted to irregular sampling network treatment, described in [4.3], will be presented. This idea led to a solution, which is similar to compensation approach, published eg. in [9]. Short remark and comparison with this method will follow. Stability of presented irregularly sampled control system will be discussed using the same idea, as described with compensation approach in [9], using stability condition developed in [11].

5.2.1 Irregularly sampled control system composition

Principle of irregularly sampled distributed control system has been described in [4.3]. Sensor and controller nodes must be able to send fresh values upon network release. Empirically, intersample distances must satisfy some requirements for system to be stable and operate well. It seems they should be shorter or equal than maximal allowed value, or to lie in prescribed interval.

At this point, we will establish a node implementation layer, which will service network events and assure prescribed intersample distances – event service layer. This layer could run within interrupt or operating-system event handlers, regarding events from network and timer. It should provide required maximal and minimal sampling rate – the required constraints. This layer would be present in sensor and controller node. At sensor node, event service layer would ask at each particular instant new measured value. Now, there remains a question of controller implementation over the event service layer.

Following picture shows schematically implementation structure of controller node. Controller itself is fed by samples $y[m]$, received from sensor at time $t_s[m]$, both values provided by event service layer. Next, it is asked to provide anytime a command value $u(t)$, sent time to time (at time instant $t_c[k]$) by event service layer as $u[k]$, and the time value $t_c[k]$ is fed back to the controller also. (Note: $y[m]$, $t_s[m]$, $u[k]$ and $t_c[k]$ are discrete values, where $m$, $k$ are integer sample numbers; different letters has been used to emphasize, that these samples need not to have any direct relation.)

Controller and sensor nodes with irregular sampling are illustrated in following schematic of closed control loop. Irregularly sampled command value is put by controller-actuator messages at actuator zero-order hold.
Plant output is irregularly sampled at sensor and fed back to controller. Sampling instants of both sampling elements are imparted to the controller.

control loop with irregular sampling at two points

5.2.2 Irregularly sampled controller design

We have to find a controller, which will receive samples measured by sensor and sampling time values of both measurements and actuation, and which will deliver a command value is expected at each actuation sampling instant. The problem is, we need to compute command value anytime, even at instants, when measurement is not momentarily performed (available). Command value should result from system state (plant state in relation to reference value), thus we need to estimate the system state anytime, when the command value is needed.

Similar problem appears, when controlling system with known constant transport delay, solved often using Smith predictor [2]. Such controller contains a plant model, estimating output of plant without the delay, which estimate is used to compute command value by arbitrary controller. Delayed measured real output value is compared to estimate and remaining error is then corrected. This idea is known as internal model principle. Our task of irregularly sampled controller is very similar. Instead of delay, we have an irregularly sampled plant input and output values.

Let us have an internal model of plant inside the controller. Model is actuated by irregularly sampled command values, held by zero-order hold – by a step sequence, which is known to controller and described as \((u[k], t_c[k])\). Output \(\hat{y}\), or a state vector\(^{24}\) estimate is fed to the controller core, computing command value from this estimate. After arrival of each measurement \((y[m], t_s[m])\), the state of model is updated to eliminate difference between estimate and reality, caused by model imperfection\(^{25}\). Command value is thus computed from system state estimate, which approaches real system state, if appropriate sensor sampling and model is used. Between sensor sampling instants, controller works in open-loop, as if controlling internal model only.

Described internal model idea is well-known in ordinary discrete as well as continuous-time control theory, and it is called a state observer [1]. Our irregularly sampled observer should provide a state estimate anytime, so it can be regarded as continuous-time observer. On the other hand, both inputs of irregularly

\(^{24}\) in LTI systems domain, the output is a linear combination of state vector, so later only a state (vector) will be discussed

\(^{25}\) it is possible to update model parameters (coefficients) as well – such strategy would belong to field of adaptive control and has not been examined in this work
sampled observer – command and plant output values – are discrete (sampled), as with discrete-time observer. Controller containing such irregularly sampled observer (model) is illustrated in following figure.

controller based on irregularly sampled observer

5.2.2.1 Irregularly sampled observer

First, we will discuss, if a design of continuous-time state observer with irregularly sampled inputs is possible. Consider a plant, described in continuous-time state space by equation:

\[ \dot{x}(t) = Ax(t) + Bu(t) \]

with square matrix \( A \) and one-column matrix (vector) \( B \). Plant and observer are actuated by piecewise constant signal \( u(t) \). This signal is equal to sample value \( u = u[k] \) during time interval \( [t_c[k], t_c[k+1]) \), \( T[k] = t_c[k+1] - t_c[k] \), at \( k \)-th actuation step. Evolution of controlled system state at instants \( t_c[k] \) thus can be described as

\[
\bar{x}[k+1] = \Phi(T[k])\bar{x}[k] + \Gamma(T[k])u[k]
\]

where square matrix \( \Phi \) and one-column matrix \( \Gamma \)

\[
\Phi(T) = e^{AT} \quad \Gamma(T) = \int_0^T e^{At}dB
\]

are known as discretization of continuous system \( A, B \) for discrete-time period \( T[k] \) in terms of (regularly sampled) ordinary discrete control [1]. System state in arbitrary time \( t \), where actuation value \( u(t) \) is known, is by analogy equal to

\[
\bar{x}(t) = \Phi(t - t_c[k])\bar{x}[k] + \Gamma(t - t_c[k])u[k]
\]

where \( k \) is the greatest \( k \), for which \( t_c[k] \leq t \). This equation gives us a state estimate in any arbitrary time during known actuation, if the state \( \bar{x}[k] \) is known.

\[\text{26 where it makes a difference, single-input single-output system will be considered}\]
Let \( n \) be a system order and system output to be computed as \( y(t) = C\bar{x}(t) \) by one-row matrix (transposed vector) \( C \). If we have received \( n \) measurements of output value, \( y[m_1, \ldots, m_n] \), from sensor, we can write system of \( n \) linear equations in form

\[
y[m] = C(\Phi(t_s[m] - t_c[k_m])\bar{x}[k_m] + \Gamma(t_s[m] - t_c[k_m])u[k_m])
\]

where \( k_m \) is index corresponding to particular time \( t_s[m] \) (greatest \( k_m, t_c[k_m] < t_s[m] \)). All state vectors can be substituted by their evolution from some chosen state \( \bar{x}[k_m] \) (either in future or in past, but must be covered by known actuation value \( u(t) \)). As a result, we have a linear equation system with known measurements \( y[m_1, \ldots, m_n] \), actuated values \( u[k] \) and sampling instants \( t_s, t_c \) and unknown state vector \( \bar{x}[k_m] \).

Solving this linear equation system at each measurement arrival computes a new state estimate. By this way of direct computation, a deadbeat observer, in terms of ordinary discrete control, is obtained [1]. However, in practice, deadbeat observer is not recommended for use, especially for its sensitivity to measurement errors. A dynamical observer with slower state adaptation should be used instead. The deadbeat observer has been used here as a demonstration of irregularly sampled state observation possibility.

5.2.2.2 Treatment of optional variable delays

Primarily, we consider irregularly sampled control system without variable delays. The main reason is, that in our distributed control system, the time of controller-actuator message sending is never known much long before. However, optional delay between measurement and sensor-controller message sending can be measured and theoretically also time to next possibility of controller-actuator message sending can be determined (e.g. if message of other node is in progress and we can read its datalength field before transmission is finished).

Both these delays can be regarded as one delay \( \tau_k \), a time distance between last measurement \( t_s \) and current next actuation time \( t_c \). However, introduction of this delay will not influence anything on the irregularly sampled observer operation. As the proposed controller computes its output from observed state only, the delay will not influence the design as well. For such type of controllers, it means no loss of generality to talk only about irregular sampling without delays.

5.2.2.3 Controller with irregularly sampled static state feedback

Irregular sampling according to desired constraints using sensor-controller and controller-actuator messages has been entrusted to event service layer, continuous-time state estimation has been solved by irregularly sampled observer and the remaining question is to determine the command value by controller feedback core.

It has been decided for our design, that command value will depend on estimated state only. With ordinary discrete controllers, as well as with continuous-time controllers, the state feedback is often used – command value is a scalar product of constant vector \(-L\) (row) and state vector (column), \( u(t) = -L\bar{x} \). \( L \) is chosen to lead state vector to zero during state evolution. By this, the less general problem of regulation to the origin\(^\text{27}\) is solved, and thanks to system linearity, particularly scaled reference \( r \) is then simply added to command signal, \( u(t) = Rr - L\bar{x} \). In following, regulation to the origin with reference value omitted will be discussed mostly to achieve simplicity.

To get a linear controller, \( L \) should not depend on \( \bar{x} \). To get a time invariant system, it also should not depend on time (however, there is no reason for it). Moreover, we have to choose, if \( L \) will depend on sampling information (intersample distances) or not. For maximal simplicity, controller with constant (static) controller vector \( L \) is proposed. Actuated command value is thus \( u[k] = u(t_c[k]) = -L\bar{x}(t_c[k]) \).

\(^{27}\) zero vector
To analyze operation of controller with irregularly sampled state feedback, we first assume, that observer state estimation is sufficiently precise. We assume, we have anytime in the controller an estimate, “copy”, of controlled system state, $\bar{x}$. Each actuation sampling instant $t_c$, new computed command value is held at plant input. Evolution of system state at actuation sampling instants thus can be described as

$$\bar{x}[k+1] = \Phi(T[k]) \bar{x}[k] + \Gamma(T[k])(-L\bar{x}[k])$$

If the sampling period would be constant, matrices $\Phi, \Gamma$ and resulting closed-loop matrix $M = \Phi - \Gamma L$ will be constant, reducing the case to regularly sampled ordinary discrete control system. This is natural and it agrees with above statement, that $\Phi, \Gamma$ matrices are equal to matrices, resulting from system discretization for given period.

Our goal is to ensure stability of system, using controller with irregularly sampled state feedback. We need to show, that state $\bar{x}[k]$ evolution will converge to desired position, in this simplified case to the origin.

In connection with note about compensation approach, a semi-empirical guide will be shown to stability test of proposed controller with given $L$, plant $A, B$ and intersample distances set $T[k] \in T$.

### 5.2.3 Note about “compensation approach”

When searching for state-of-the-art in design of controllers adapted for distributed systems, one approach, almost the same as irregularly sampled controller presented above, has been encountered. It is called a compensation approach and appears in work [9]. However, two distinct controller principles are covered under this name:

- adjustment of classical controller (PID) according to particular sensor intersample distance – eg. difference action is divided by this distance, integral action is multiplied by it; described in [10], [9];
- controller with irregularly sampled observer and variable state feedback – irregular sampling and known variable delays are handled by this controller, using the same idea of observer and state feedback vector is a function of particular intersample distances and known delays; described in [9]; this is the idea, which is close to our controller design and which will be uniquely called a compensation approach in following text.

Basic idea of both controllers, irregularly sampled with static state feedback, as well as compensation approach, is similar – both use an observer, estimating system state in arbitrary time, based on discrete irregularly sampled data, and both use a state feedback. The difference is, that compensation approach works with time intervals between (last) measurement and actuation separately, calling them to be delays ($\tau[k]$), and that compensation approach has a state feedback vector as function of particular intersample distance and mentioned delay, $L = L(T[k], \tau[k])$. To determine value of this vector (and also of similar vector $K$, influencing observer dynamics), the pole placement problem is solved for each $(T[k], \tau[k])$ pair. Noticeable differences between irregularly sampled controller with static state feedback and compensation approach are:

- feedback vector $L$ is a function of sampling in compensation approach, $L = L(T[k], \tau[k])$; from this point of view, the compensation approach is more general, because in static state feedback irregularly sampled controller the $L$ is constant;
- in compensation approach, command value is sampled as a consequence of measured output sample arrival – according to code excerpt, published in [9], there is a “1:1” relation between sensed and actuated samples in compensation approach; in proposed irregularly sampled controller with static state feedback, input and output samples are completely asynchronous, allowing sampling to be in arbitrary
Both these controllers are subset of more general controller, which has no direct relation between system output and command value sampling (asynchronous sampling), and where the state feedback vector is a function of sampling \( T[k] \), optionally \( \tau[k] \). In our proposal of irregularly sampled controller with static state feedback, constant feedback vector was chosen, to allow simpler design and stability discussion procedure.

For our irregularly sampled controller, the most interesting part of work [9] about compensation approach is a stability condition (proof), took from article [11]. Subset of this condition will be used in next section for stability discussion of proposed irregularly sampled controller with static state feedback.

5.2.4 Stability of irregularly sampled state feedback controller

Sufficiently precise state estimation by observer has been assumed in [\( \rightarrow \cdot 5.2.2.3 \)]. Then, the system is asymptotically stable if and only if its state \( \vec{x} \) will converge to origin for any infinite sequence of actuation intersample distances \( T[0], \ldots T[\infty] \) from any initial state \( \vec{x}[0] \):

\[
\lim_{k \to \infty} \vec{x}[k] = \lim_{k \to \infty} M(T[k])M(T[k-1]) \ldots M(T[0])\vec{x}[0] = \vec{o}
\]

where \( M(T) = \Phi(T) - \Gamma(T)L \) is a closed-loop matrix for step of given duration \( T \). Convergence must be assured for any sequence of possible matrices. Set of all possible matrices \( \mathcal{M} \) for given controlled system is determined by set of possible intersample distances \( \mathcal{T} \), because feedback \( L \) is said to be constant. If \( \mathcal{T} \) is a continuous interval, both sets will be infinite.

As mentioned in [9], [11] gives a sufficient and necessary condition of asymptotical stability for system with given matrix set \( \mathcal{M} \). The condition is:

\[
\exists P \in \mathbb{R}^{n \times n}, P > 0, \exists k \in \mathbb{N}, \quad M^T P M - P < 0, \quad \forall M \in \mathcal{M}^k
\]

where \( P, M \) are matrices of dimension \( n \), relational operators \( >, < \) denotes matrix inequalities in sense of positive (negative) definiteness\(^{28}\), and \( \mathcal{M}^k \) denotes a set of all possible products of \( k \) matrices taken from \( \mathcal{M} \).

In [9], a simplified sufficient although not necessary condition is mentioned, taking only members of tested \( \mathcal{M} \) set and not their products:

\[
\exists P \in \mathbb{R}^{n \times n}, P > 0, \quad M^T P M - P < 0, \quad \forall M \in \mathcal{M}
\]

This condition can be translated to human language as “exists a coordinate transform of system state vector, in which norm of the state vector decreases after each multiplication by any matrix from \( \mathcal{M} \)”. It means, in each regulation step the size of state vector diminishes, in some coordinate transformation. In earlier necessary and sufficient condition, the decrease is not required in each step, but in each \( k \) steps at least. In following, we will discuss only the simpler \( \exists P > 0, M^T P M - P < 0, \forall M \in \mathcal{M} \) sufficient condition.

Note, that for single matrix \( \mathcal{M} = \{M\} \), both conditions are equivalent to \( M \) having all eigenvalues inside unit disc\(^{29}\), \( \exists P > 0, M^T P M - P < 0 \Leftrightarrow \rho(M) < 1 \), what is in accord with regularly sampled discrete system stability condition. In case with multiple matrices in \( \mathcal{M} \), eigenvalues of each matrix \( M \in \mathcal{M} \) must lie inside unit disc \( \rho(M) < 1 \) as a necessary condition for overall asymptotic stability.

\[
^{28} \text{matrix } A \text{ is positive definite } A > 0 \Leftrightarrow \vec{u}^T A\vec{u} > 0, \forall \vec{u} \in \mathbb{R}^n
\]

\[
^{29} \rho(M) \text{ means spectral radius of matrix, ie. the greatest eigenvalue absolute}
\]
5.2.5 Irregularly sampled static state feedback controller design

In previous, structure of controller with irregularly sampled static feedback has been established and condition ensuring control system asymptotical stability has been shown. Choice of state feedback vector $L$, solved in ordinary discrete and continuous-time control by pole-placement, will be discussed in following, together with directions, how to check resulting system stability.

The question is, how to choose state feedback vector $L$ to get reasonable control performance and assure stability within required range of sampling periods $T$. In ordinary discrete and continuous-time feedback design, $L$ is computed by pole (eigenvalue) placement of closed-loop matrix:

- in continuous-time, closed loop state evolution is described as $\dot{x} = (A+BL)x$; eigenvalues of $(A+BL)$ are chosen to have their real parts negative $\forall \lambda, \Re \lambda < 0$; more negative, faster the system response;
- in ordinary discrete-time, closed loop state evolution is described as $x[k+1] = (\Phi + \Gamma L)x[k]$; eigenvalues of $(\Phi + \Gamma L)$ are chosen to lie inside the unit disc $\forall \lambda, |\lambda| < 1$; closer to origin, faster the system response.

We have to select $L$ for usually infinite matrix set $\mathcal{M} = \{M| M = \Phi(T) + \Gamma(T)L, T \in \mathcal{T}\}$, moreover, it is necessary but not sufficient condition for all these matrices to have their eigenvalues inside the unit disc. The same relation as with ordinary discrete system appears here, that closer these eigenvalues (of all matrices) are to the origin, faster the overall system response is. Based on this observation, sophisticated design methods for finding optimal value of $L$ could be developed. The design could be much improved, if probabilistic distribution or probability of consequence of intersample distances $T \in \mathcal{T}$ is known. In following, we will show a very simple empirical way how to choose feedback vector, although it is in no means guaranteed to be successful in every case.

If intersample distances would be near to zero $T \to 0$, system state evolution will approach the evolution of continuous-time system. If intersample distances would be near to constant $T \to T_0$, system state evolution will approach evolution of regularly sampled discrete system. Thus, in great simplification, we can design feedback vector $L$ according to continuous-time pole placement, using $A, B$ system matrices, if treating intersample distances as negligible, or according to discrete-time pole placement, using $\Phi, \Gamma$ matrices, evaluated for chosen “representative” period $T_0$. However, this placement will not assure stability for other matrices from $\mathcal{M}$ (intersample distances from $T$), thus a stability check should be performed.

Possible stability check is by sufficient condition $\exists P > 0, M^TPM - P < 0, \forall M \in \mathcal{M}$. For a finite set $\mathcal{M}(T)$, this condition can be tested by numerical always convergent method of solving LMI\(^{30}\), as proposed in [11], [9]. From theoretical point of view, the question is, whether $T$ is a finite set of possible time distances, or an infinite continuous interval. In practice, there will be always possible to find sufficiently large finite subset $T' \subset \mathcal{T}$ to approximate infinite $T$. If the desired $T$ and its finite approximation $T'$ are known and computational power of LMI solver is sufficient to solve resulting LMI (containing as many source equations, as is the number of $T'$, what means number of $\mathcal{M}$ elements), the question of stability is answered. If either $T$ is not given precisely or LMI solving seems to be complicated, following simple test could be performed. Note, that its success depends on given system and feedback vector and if stability in some range of $T$ is not proven by it, it does not imply unstability.

We have system matrices $A, B$ and given feedback vector $L$. We will try to determine some range $T$, where irregularly sampled closed loop will be stable. As mentioned above, a necessary condition of stability is, that for each matrix $M \in \mathcal{M}$ (for each $T \in \mathcal{T}$) its eigenvalues must lie in unit disc:

$$\forall T \in \mathcal{T}, \rho(M(T)) < 1$$

---

\(^{30}\) Linear Matrix Inequalities, see Matlab Robust Control Toolbox version 3.0.1, or previous LMI Control Toolbox
We can evaluate spectral radius of $M(T)$ for any reasonable range of $T$ and plot it into graph. Of course, only finite number of points can be plotted, but the curve will give a good view of areas, for which $T$ the stability can not be assured, and for which it is possible.

Spectral radius of four different feedback terms for the same system is plotted into following graph. Controlled system has the same structure and almost same parameters, as our experimental servomotor plant. Its statespace description is $A = \begin{pmatrix} -5 & 0 \\ 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 40 \\ 0 \end{pmatrix}$, $C = (0 \ 1)$. Plots show interval $S$ for each feedback $L$, where must every $T$ lie, otherwise the system could be unstable. For feedback $L_1 = (0.35 \ 4)$ (designed by continuous pole placement to $s_0 = -9.5 \pm 8.3516$), the required interval is $S_1 = (0, 0.136)$. For $L_2 = (0.3183 \ 3.3593)$ (designed by discrete-time pole placement to $z_0 = 0.7715 \pm 0.1635$ for sampling period $T_0 = 0.025$), $S_2 = (0, 0.149)$. For $L_3 = (0.2892 \ 2.8238)$ (discrete-time pole placement to $z_0 = 0.5684 \pm 0.2522$ for $T_0 = 0.05$), $S_3 = (0, 0.164)$. For $L_4 = (1 \ 30)$ (continuous-time pole placement to $-22.5 \pm j26.3391$), $S_4 = (0, 0.0488)$.

Note, that for $T \to 0$, spectral radius approaches one. It is because of closed-loop matrix approaches the identity. Stability for infinite sampling frequency is determined by continuous-time description as $\forall \lambda$ is eigenvalue of $(A - BL)$, $\Re \lambda < 0$. However, in our distributed system design this is out of the question.

Now we know, for which $T$ the designed $L$ should not be used, where the loop stability can not be guaranteed. If the result is unsatisfactory, we should redesign $L$, eg. using discrete pole placement with other representative $T_0$. Set (interval) $T$ of allowed $T$ must be a subset of obtained area $S \supseteq T, S = \{T|\rho(M(T)) <$
After selection of some $T$ (and finite subset $T'$ according to desired precision), LMI problem can be solved. If selection of $T$ or LMI solving is not wanted, an empirical way to find a $P$, for which $M^{T}PM - P < 0$ could be satisfied, can be tried.

As mentioned above, condition $M^{T}PM - P < 0$ is always met for single matrix $31$, if and only if $\rho(M) < 1$. Condition $M^{T}PM - P < 0$ can be easily checked for diagonalizable matrix $M = VDV^{-1}, D = \text{diag}(\lambda_1, \ldots, \lambda_n)$ by substitution $32 P = (VV^{*})^{-1}$, and $M^{T} = M^{*}$.

Now, we can choose some “representative” intersample distance $T_0$ and evaluate matrix

$$P_0 = (V_0V_0^{*})^{-1}, M(T_0) = V_0D_0V_0^{-1}$$

With this matrix $P_0$, we can check condition

$$M^{T}(T)P_0M(T) - P_0 < 0$$

by evaluating eigenvalues for each tested $T$ ($\forall \lambda$ is eigenvalue of $(M^{T}(T)P_0M(T) - P_0), \Re \lambda < 0$). By plotting maximal real parts of eigenvalues for all interesting $T \in S$, subset $T$ is determined, where above condition is satisfied with matrix $P_0$. For this subset, asymptotical stability is assured. But, some other matrix $P$ may be found, covering larger subset of $S$. If resulting set (interval) $T$, for which asymptotical stability is proven by matrix $P_0$, is satisfactory, the design is finished and desired constraints for $T$ are determined. If $T$ is unsatisfactory, finding another $P$ should be tried, eg. by choosing another representative $T_0$.

In following graph, maximal real parts of $M^{T}(T)P_0M(T) - P_0$ eigenvalues are plotted for the same feedback term $L = (0.35 4)$ and four different test matrices $P_0$, assuring stability for different intervals $T$. Matrix $P_0$ evaluated for “representative” period $T_1 = 0.025$ assures stability for interval $T_1 = (0, 0.119)$ for given feedback $L$, $P_0$ evaluated for $T_2 = 0.005$ assures stability for $T_2 = (0, 0.112), P_0$ evaluated for $T_3 = 0.05$ assures stability for $T_3 = (0, 0.127)$ and $P_0$ evaluated for $T_4 = 0.19$ assures stability for $T_4 = (0, 0.127)$. As a result, we can say, that statical feedback $L = (0.35 4)$ controller will be asymptotically stable, if actuating intersample time distances $T \in T_4 = (0, 0.127)$.

\footnote{has been checked for real diagonalizable matrices}

\footnote{$V^{*}$ designates adjoint or conjugate transpose matrix, ie. matrix transpose with complex conjugate members}
Simulated transient response of described controller is plotted into following picture. Observer model was $A =\begin{pmatrix} -5 & 0 \\ 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 40 \\ 0 \end{pmatrix}$, $C = (0 \ 1)$, while controlled system was slightly different to simulate model imperfection: $A = \begin{pmatrix} -4 & 0 \\ 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 38 \\ 0 \end{pmatrix}$, $C = (0 \ 1)$. Feedback $L = (0.35 \ 4)$ has been used and $T < 0.05$ was guaranteed by event service layer, implemented by dynamical priority deadline $[\rightarrow 6.8]$. Same traffic generation parameters were used as in $[\leftarrow 4.2.1]$. Dynamical observer is computed according to

\[
\hat{x}_m[t] = \Phi(t - t_c[k])\hat{x}_m[t_c[k]] + \Gamma(t - t_c[k])u[k] \\
\hat{x}[m] = \hat{x}[m] + K(y[m] - C\hat{x}[m]) \\
\hat{x}(t) = \Phi(t - t_c[k])\hat{x}(t_c[k]) + \Gamma(t - t_c[k])u[k]
\]

- it is an analogy of current observer; observer vector $K = \begin{pmatrix} 36 \\ 1 \end{pmatrix}$ was used. In the graph, output signal (measured position) and its sampling is shown in upper part. Position and its derivative (speed), forms a state vector of controlled system. Speed component is plotted in the middle. Solid line represents real value and dashed corresponding observed estimate (however, in case of position the difference is not visible). At the bottom, evolution of command value and its sampling is plotted.
irregularly sampled control system response
6 Dynamical priorities

This section deals with improvement of ordinary discrete controller operation in distributed environment, using dynamical assignment of message priorities.

6.1 Motivation

By assignment of particularly high priority to control system messages in comparison to priorities of environment messages, negative influence of network to control performance and stability could be eliminated. If sensor and controller-actuator messages have the highest priority on the network, problem of distributed control is reduced to ordinary discrete control problem, because network induced variable delays disappear (if neglecting effect of priority inversion). However, it seems to be desirable to give high priority to other tasks (from our point of view, environment) also, ie. to share not only the network capacity, but also “short response time”.

Furthermore, in many control tasks, it seems, there are certain system states, where a precise operation of loop is not so mandatory, states, when the feedback loop is “idle”, or not so sensitive to delays or erasures. The idea of dynamical priorities is, that we can lower the priority of control loop messages in some states and to get high priority again while we need it.

This will shorten network response time for environment tasks, if control system is in state, when low network priority is allowed. The assumption is, the system and regulation requirements have such states.

6.2 Message priority implementation

Strategy of dynamical priority assignment has been implemented using CAN hardware. For each controller-actuator message, a priority is computed in controller node and set for the message transmission. On CAN, the priority value, contained in arbitration of identifier field of CAN message, is an integer number, serving also as unique message identifier. Priorities are compared arithmetically, but with inverted meaning of digits (bits), ie. zero is treated as “higher” than one.

We want not to influence priority competition between controller and environment messages by identification, thus we must assign the least significant bits for the purpose of message identification and put the dynamically computed priority into more significant bits. The identifier field has been divided into three subfields:

![Diagram of CAN identifier field](image)

---

33 anyway, we will call a priority with higher precedence as a “higher priority”, although it is represented by lesser number
The most significant part is a statical priority, allowing to set relative position of environment and controller priority intervals. The middle part is dedicated for computed dynamical priority and together with statical part it holds generated environment priorities. The least part is intended for message identification, to distinguish between three types of messages – sensor, actuator and environment. In our implementation, 11 bit field has been divided as follows:

<table>
<thead>
<tr>
<th>S</th>
<th>D</th>
<th>I</th>
</tr>
</thead>
</table>

Ie. to 1 bit of statical part (S), 8 bits for dynamical assignment (D) and 2 bits for identification (I). Sensor-controller messages have identification pair equal to \(0\) and \(1\), controller-actuator messages \(0\) and \(1\), and environment messages \(1\) and \(1\). Messages with dynamically assigned priorities have the statical field constantly set to \(1\). Priority range for controller-actuator messages is between \(0x401\) and \(0x7fd\) inclusive with granularity of 8 bits (256 distinct values).

Following text will deal only with the statical and dynamical fields, constituting a priority number. Note, that priorities causing higher precedence will be treated as being greater number (\(0x3f0\) is greater than \(0x400\)).

6.3 Sensor-controller message priority

Node, which has usually the most complete information about system state, is the controller node, due to presence of reference value. Thus, there is no problem with assignment of priority to an controller-actuator message. But there remains a question of determining sensor-controller message priority:

- sensor-controller message with statical priority (high) – the disadvantage is network priority occupation all the time; advantage is simplicity, good periodic timing at controller node and elimination of sensor sample erasures (see \[\text{[---]}\]);
- sensor-controller message priority learned from last actuation message – by principle of CSMA network, all nodes can read all messages, what allows sensor node to read priority value from actuation message; also the controller can put some additional field to message (or even additional message, which seems to be impractical), to tell sensor node its current message priority.

The approach of sensor-controller message priority learned from controller node seems to be reasonable. In our experiments, we have tested only the statical priority of the sensor-controller messages with dynamical assignment on controller-actuator messages only.

6.4 Distribution of environment message priorities

In order to design distributed control systems, working on networks loaded by environment traffic, we will try to describe environment traffic in a very simple and general way. This description will allow us construction of simple traffic generator.

Network environment sends its messages with priorities within some range. Rate and consequence of occurrence of messages having certain priority depends on the environment and usually is not known to our
task in other than probabilistic fashion. At least the maximal rate of messages must be known, to be sure, that network capacity is sufficient for all connected systems.

An environment message can be described in simplification as a random or periodic message of average rate $r_e \in (0, 1)$, what means, this message of duration $L$ will appear in time $t \to \infty$ approximately $r_e t / L$ times. The rate represents a fraction of overall network capacity, required by certain kind of messages, but it tells nothing about their short-time behaviour, ie. if they appear periodically (widely spaced), in groups, or randomly (see \[ \rightarrow 6.5 \]).

For environment messages described this way, we can determine the rate for each given priority. We can define a function $d_e : (0, p_{max}) \to (0, 1)$, $\int_0^{p_{max}} d_e(p) dp \to 1$ assigning a rate $r_e = d_e(p)$ to message of priority $p$ (where $0, p_{max}$ is the network priority range). This function has meaning of probabilistic distribution – if we have to send message of priority $p_c$, there is for each moment a probability $P = \int_{p_c}^{p_{max}} d_e(p) dp$, that in the moment a message with higher priority is being transferred.

If the distribution $d_e$ is some general function, the problem can be translated to equivalent case with uniform distribution $d_e'(p) = \text{const.}, p \in (p_1, p_2)$, when $d_e(p) = 0, p \notin (p_1, p_2)$, when our control system message priority $p_c$ is “shaped” by inverse function of integral of the distribution (theoretically: $p_c = p_1 + (p_2 - p_1) F^{-1}(p_c), p_c \in (p_1, p_2), F(p) = \int_0^{p_{max}} d_e(p) dp$). Without loss of generality, we can expect network with uniform priorities distribution in some range and particularly take into account appropriate priority shaping function in control system nodes.

### 6.5 Environment traffic simulation

For control system simulation and experimental measurements, the network load generator, simulating environment traffic, is needed. As observed above, generator of messages with uniformly distibuted priorities between $(p_1, p_2)$ and total message rate $r_e$ is sufficient.

Average rate has been assumed to be constant, but there remains a question of short-time behaviour. In the regular case, generated messages are put into a FIFO with constant period $T_e = L / r_e$. In the opposite, random case, a random variable $X \in (0, 1)$ is evaluated periodically with period $L$, and a message is put into the FIFO, if $X < r_e$.

To cover both extreme behaviours and to allow even some interpolation between them, algorithm using exponential forgetting was used. Variable, containing current network load estimate, is in each step (period $L$) compared to desired network load, and if smaller, new message is put into the FIFO. Network load estimate is result of first order integration $y[k] = (1 - \alpha)y[k - 1] + \alpha x[k - 1]$, where $x[k] \in \{0, 1\}$ means, whether the message has or has not been sent in $k$-th step. Some amount of uncertainty can be added to comparison of estimated and desired load, which allows to set a compromise between absolutely regular and absolutely random behaviour.

Following chronographs show three events of artificial traffic generation, all three with the same desired load of 87% and with parameter of regularity ($r$) set to different values. Traffic generator node is the only network user during these events, curve is in top position while generator sends messages, and bottom, when the network is idle.
environment traffic generation with the same rate (87%) and varying regularity

6.5.1 Implementation

In C167 microcontroller traffic generator for CAN, algorithm with exponential forgetting has been implemented, with hardcoded constant $\alpha = 1/256 = 0.039$ and user adjustable regularity and desired load on the fly, using serial terminal. Messages have constant length of 47 bits at link layer, i.e. zero length data field on CAN 1.0, and timer runs at corresponding period $T = N/c[s; \text{bit, bit s}^{-1}], N = 47\text{bit}$, where $N$ is message length and $c$ network capacity.

User enters generated priority range, priority bitmasks, resulting in generated priority \texttt{priority = mask_or | (mask_and & random(p_min, p_max))}, where \texttt{mask_or, mask_and} are user entered bitmasks, \texttt{and} \texttt{&} are bitwise OR and AND operators, respectively, and \texttt{random(p_min, p_max)} is a function, generating random integer within user entered range $\langle p_{\text{min}}, p_{\text{max}} \rangle$. The regularity parameter is in range $\langle 0, 1 \rangle$, where 0 is completely random and 1 regular. To get somewhere “in the middle” between regular and random behaviour, it should be very close to 1. The desired load parameter is in percent of network capacity, but effectively it is not precise, because of \textit{bit stuffing}[	extsection 6.5.2]. Example of terminal session with realized traffic generator follows.

```
-> c
Min. priority (ID) number (0..0x7fd): 0x380
Max. priority (ID) number (0x380..0x7fd): 0x420
Priority AND mask [0xfff]): 0x7fc
Priority OR mask [0x000]): 2

CAN speed: 50kb/s
Network load: 0.000
Regularity: 1.0000
Current load: 0.389
Priority
  range: 0x380 -- 0x420
  AND mask: 0x7fc
  OR mask: 0x002
```
In TrueTime, the same traffic generation algorithm has been used. However, dynamical priorities have not
been used within TrueTime, and also the message identification is independent on message priority. True-
Time traffic generator thus generated statically prioritized messages at given rate (capacity fraction) and
regularity. TrueTime M-file traffic generator code example follows. This procedure is called as periodic event
by a TrueTime kernel. Variable \texttt{data.cl} holds current generated load estimate, \texttt{data.alpha} is exponential
forgetting coefficient, \texttt{data.BW} is desired generated load in \((0, 1)\) and \texttt{data.reg} is regularity.

\begin{verbatim}
data.cl = (1-data.alpha)*data.cl;
x = data.reg*data.cl + (1-data.reg)*rand;
if x<data.BW,
data.cl = data.cl+data.alpha;
ttSendMsg(4, 0, 50, 003);
end
\end{verbatim}

6.5.2 Bit stuffing difficulty

Actually, there is a problem to generate precise amount (capacity fraction) of network load on a CAN
network. Each message, composed at link layer of data field, identifier, CRC\(^{35}\) and other protocol bits, is
subject to bit stuffing, i.e. addition of inverse valued bits after each 5 consecutive bits of the same value.
This means, that messages are at physical layer always longer, than at link layer, and that their length vary
depending on content, including the identifier (priority). For precise network load simulation, the calculation
of CRC and number of stuffed bits should be done in generator. I have not implemented this.

On the one hand, the difference of message length is not so mandatory, on the other hand, if we generate
eg. 47bits long messages for calculated 93% of network load and neglect bit stuffing, it gives 100% load instead
(measured on real system with exponentially forgetting traffic generator in C167 node).

6.6 Message priority computation

Priority of actuation message (optionally also of the next sensor-controller message) is computed in controller
node and should be based on control system state. Control system state is a state of controlled system in

\(^{35}\) Cyclic Redundancy Check for transmission error detection

38
relation to reference signal. A simple value describing the state, is difference between reference and actual output value – the control error.

In presented regulation task, the goal is to force the error and its derivatives to zero, which we call the steady state. Thus, we tried to base priority calculation on the error. Input to the priority calculation in each step is an error, in general unlimited \( e \in (-\infty, \infty) \) (in positional motor control it is only within range of circle), and output is a message priority \((0, p_{\text{max}})\).

### 6.6.1 Error proportional priority

We can compute priority from the error by some simple function, mapping error to priority space \( p = f(e) \). In every linear and most nonlinear controllers, the sign of error makes no difference in dynamical behaviour, so the priority can be a function of \(|e|\).

In our experimental realization, a saturation-like function has been used \((p = f(|e|), |e| < e_s : f(x) = p_0 + p_g/e_s|e|, |e| \geq e_s : f(x) = p_0 + p_g)\). Beside the function gain \(p_g/e_s\) and saturation point, it is very important to determine sign of \(p_g\), i.e. whether the priority is positively or negatively proportional to error.

![saturation-like error to priority mapping functions](image)

If control system does not change any value in steady state, the feedback is not necessary in such state, until new disturbance or reference step arrives. This is not true for large set of systems, e.g. oscillating switched actuators, but for our motor control it is. This fact suggests use of positive proportionality (bigger error – higher priority), called later a positive approach.

On the other hand, from practice with saturation-limited systems and optimal control, it is known to work well, if the precision of regulation is increased after initial coarse drive, when the steady state is approached. It means, after reference or disturbance step, the plant is driven in particular direction, but without need of precise calculation – and when its output (state) trajectory gets closer to desired value, control loop takes its regulation work and refines the resulting value. This experience suggests use of negative approach (smaller error – higher priority).

Both of these approaches were tested in real experiment with motor control. Upper part of following graph shows transient responses of systems, using positive (thinner line) and negative (thicker line) proportionality of priority value, plotted with corresponding line widths, to error absolute. Note: in time of experiment, different saturation-like function for negative proportionality has been used, than described above. During measurement of this experiment, the negative function was \(p = f(e_{\text{max}} - |e|)\), where \(f(x)\) is positively proportional saturation-like function, as described above.

39
The observation is, that negative approach gives much better performance than positive one. This means, it remains stable with network load, where positive one is almost always unstable, and that with the same network load it has shorter stabilization time. Of course, results depend heavily on the priority function selection (saturation point), but this belongs to more general and the most important problem of solution quality criteria selection [6.7].

Negative approach can suffer by a delay immediately after error step or while the error is big, what has been visible on some of response plots. But these delays were negligible in comparison with duration of system stabilization process. Using positive approach, node yields its priority each time the system output approaches reference value. Because of system inertia, it does not mean, that the system is stabilized, if its output is close to desired value. Generally, the system state is a multidimensional (vector) variable, and output value is its scalar product with some constant, thus immediate value of the error equal to zero can correspond to many states, from which only one is really a steady one. This is an explanation, why system with positive approach got often unstable, oscillatory response – it lost the control in a moment, when error was zero, but velocity high.

The negative approach offers good system stabilization, but at a cost, which seems to be unacceptable – it takes away the highest priority during whole steady state, what is not our objective ([6.1]). Roughly speaking, while nothing is happening, controller occupies prioritized network access. As we assumed above at least for some set of systems, including our plant, the steady state should be a point, where precise feedback operation is not so needed, at least until disturbance or step arrival.
Conclusion is, that some more sophisticated mechanism for computing of priority is needed. Our vague specification is:
- give lowest priority in the steady state
- give high priority while stabilizing
- immediately after reference step, high priority is not necessary

6.6.2 Priority proportional to filtered error

These statements suggest, we need to have a memory in priority computing algorithm, to distinguish between small error in steady state and small error during stabilization stage. Presence of memory means, the error is entering some dynamical system, on whose output is a priority.

In experiment, first order lowpass filter $F(z) = G(1 - \beta)z/(z - \beta)$ (the same structure, as in exponential forgetting \cite{6.5}) with time constant $\beta = 0.95$ and gain $G = 4$, chosen empirically to cover duration of stabilization stage, has been implemented. The results are better, than with proportional-only (nonfiltered) positive approach. Following graphs shows transient responses and evolution of priorities for proportional-only and filtered error computations. In both cases, the same regular traffic generation and saturation point $e_s = 0.24$ has been used.

Following graphs show comparison between experimental transient responses of systems, using priorities positively proportional to error absolute (thinner line), or error absolute filtered (thicker line) in the upper part of graph. Evolution of particularly computed priority values is plotted in the lower part of graph, again with corresponding line widths.
filtered and nonfiltered error based priority comparison (traffic load 92% with priorities in 0x400..0x480)
From measured responses, it can be seen, that filtered error approach gives slightly better control performance. It usually stabilizes in shorter time with smaller overshoot in case of particularly big reference steps. It has the same poor performance during small reference steps, as the proportional-only approach, because the error is not so big to give reasonable priority.

At certain transient, a very little delay can be observed, when using filtered error strategy, in comparison with unfiltered error absolute. This is in agreement with theory (dynamical error filter rise time). Again, it is negligible in comparison with duration of stabilization stage.

### 6.7 Problem of solution-quality criteria

The main problem of finding a dynamical priority solution is to have a criterion to determine a cost of network priority occupation and control performance. Without specification of this criteria, it is impossible to choose dynamical priority strategy, because it is not known, if it is eg. better to have a twice shorter response to given transient with ten-times longer occupation of maximal network priority.

Such criterion has not been specified for our work, because there was nothing said about requirements or cost of network environment messages transport. Above, I am only guessing, which priority assignment strategy seems vaguely to be more suitable.
6.8 Dynamical priorities with controller adapted for distributed system

As a consequence of presented irregularly sampled controller design [\textsection 5.2.5], requirement of having all actuation intersample time distances in some range has been stated. It is required for controller-actuator (and empirically also for sensor-controller, although it has not been derived mathematically) message, to be sent in some time interval after the moment, when previous message has been sent. Mainly, it should be assured, that the message will be sent \textit{at least} until some limit \textit{deadline}.

Another use of dynamical priority assignment can be for implementation of such deadline. Sensor or controller node can try sending of message first with low priority in some initial time distance after previously sent message. If network is free, the message is sent. If not, priority will be increased in each new try, and if network will be still occupied by more prioritized messages, the highest priority will be used at the last possible time, at the deadline.

This way to assure required timing for irregularly sampled controller has been used in irregularly sampled control system simulations for this work. However, if there exists some other tasks with higher than maximally assignable priority, duration of their messages should be accounted into deadline and their maximal number of occurrence in given time interval must be also assured.
7 Conclusion

- From different treatments of computer network inside a control loop, approach of FIFO-buffering has been deprecated in comparison with simple value-replacement strategy [←4.2.1].
- Networked loop treatment as a channel with irregular sampling and known measurable delays has been shown [←4.3].
- Controller design, based on irregularly sampled controlled system input and output, has been proposed, allowing independent (asynchronous) non-uniform sampling of input and output values [←5.2.2]. This controller has been compared with almost same idea of compensation approach [←5.2.3].
- Irregularly sampled controller design with static state feedback has been described together with empirical guide to stability proof together with timing constraints determination [←5.2.5].
- Dynamical CSMA/AMP priority assignment has been implemented.
- Improvement of ordinary discrete controller performance in distributed control system using dynamical priority assignment has been tested. Difference between control performance using different priority assignment strategies has been observed under very heavy network load only, near network overfill boundary.
- It has been experimentally shown, that dynamical priorities can be used in distributed system with ordinary discrete controller to get better control performance, while letting network priority to other applications during control system steady-state. However, stability can not be assured by dynamical priority assignment, where it is not guaranteed with statical priority. Also, dynamical priority assignment gives no warranty of extremal network priority occupation.
- Control performance has been optimized with respect to network priority occupation, using dynamically assigned priority based on low-pass filtered error absolute [←6.6.2].
- Another kind of dynamical priority assignment has been used to implement deadline for required timing constraints of irregularly sampled controller[←6.8].
8 Bibliography


